# Homework Assignment #2
# Threads & Concurrency

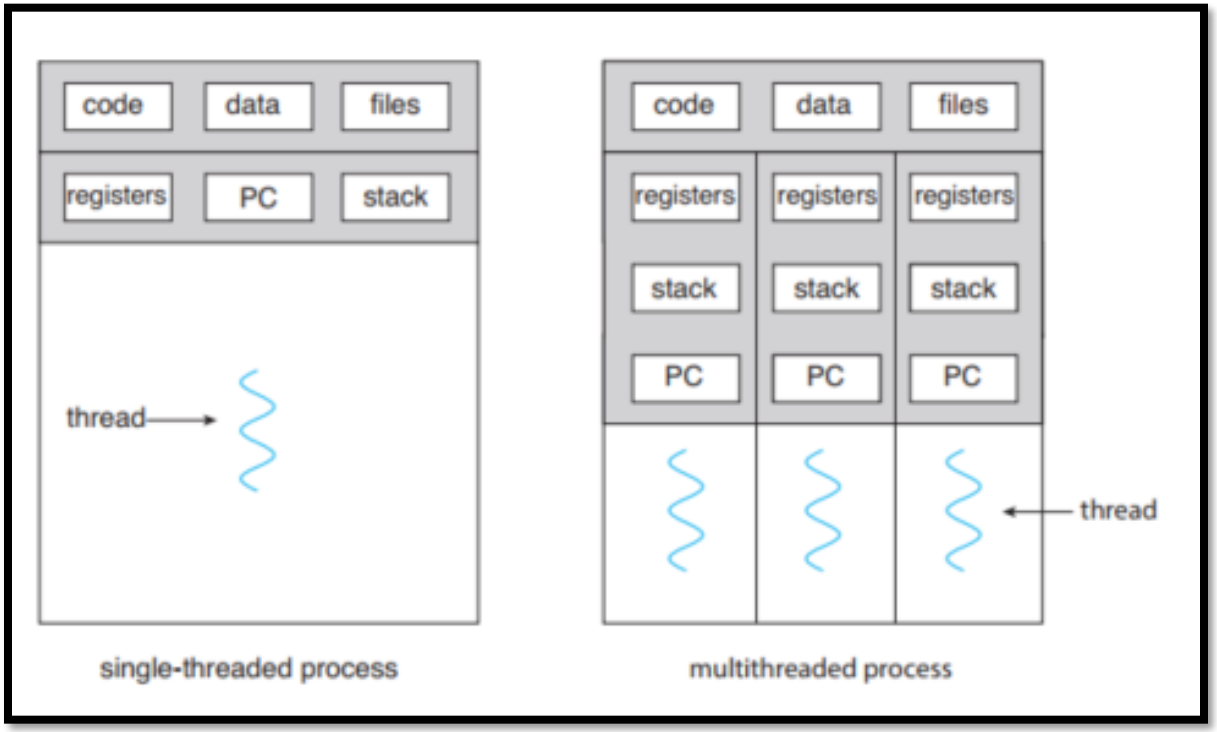
# Outline

- **Thread**
- Application Programming Interface
  - Exercise 3.20 API
  - Pthreads API
  - Other API
- Homework Assignment #2
- Race Condition
- Reference

# Thread

- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack.

- A traditional process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.



# Outline

- Thread
- **Application Programming Interface**
  - Exercise 3.20 API
  - Pthreads API
  - Other API
- Homework Assignment #2
- Race Condition
- Reference

# Exercise 3.20 API

- We have created three APIs in homework#1.
  - int allocate_map(void)
    - Creates and initializes a data structure for representing pids; returns −1 if unsuccessful, 1 if successful.
  - int allocate_pid(void)
    - Allocates and returns a pid; returns −1 if unable to allocate a pid (all pids are in use).
  - void release_pid(int pid)
    - Releases a pid

# The Pthreads API

- There is a whole set of library calls associated with threads, most of whose names start with `pthread_`
- To use these library calls, we must include the file `pthread.h`, and link with the pthread library using `–pthread`
- We will use the following APIs
  - pthread_create()
    - Create a thread
  - pthread_join()
    - Wait for a thread
  - pthread_exit()
    - Exit a thread without exiting process

# The Pthreads API (cont.)

- **pthread_create**
  - creates a thread

```
#include<pthread.h>
```

```
int pthread_create(pthread_t *thread, const pthread_att_t *attr,
void *(*start_routine)(void *), void *arg);
```

```
EX: pthread_create(&thread,NULL,PrintHello,(void*)t);
```

thread: Points to the ID of the newly created thread.

attr: An attribute object that encapsulates the attributes of a thread. NULL for the default values.

start_routine: The C routine that the thread calls when it begins execution

arg**:** The argument that is to be passed to start_routine. NULL may be used if no argument is to be passed.

# The Pthreads API (cont.)

- **pthread_join**
  - causes the caller to wait for the specified thread to exit

```
#include<pthread.h>
```

```
int pthread_join(pthread_t thread, void **value_ptr);
```

```
EX: pthread_join(thread,NULL);
```

thread: The ID of the terminating thread.

value_ptr**:** Provides a location for a pointer to the return status that the target thread passes to pthread_exit. NULL is used if the caller does not retrieve the target thread return status.

# The Pthreads API (cont.)

- **pthread_exit**
  - terminates the calling thread

```
#include<pthread.h>
```

```
int pthread_exit(void *value_ptr);
```

```
EX: pthread_exit(NULL);
```

value_ptr**:** makes the value `value_ptr` available to any successful join with the terminating thread

We can use **`pthread_exit(NULL)`** to terminate a thread.

# Thread Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// This is thread function
void *threadFunc(void *arg) {
  printf("------------------------------\n");
  printf("This is Thread Function\n");
  printf("Thread ID: %lu\n", pthread_self());   //check thread ID
  printf("Argument: %d\n", (int) arg);          // check thread's arguments

  char buf[100];
  printf("User input: ");
  scanf("%s", buf);
  printf("------------------------------\n");

  pthread_exit(buf);   // return value to line 29 as pthread_join()'s parameters
}

int main(int argc, char** argv) {
  pthread_t thread;
  int rc, t = 100;
  void *reBuf;
  rc = pthread_create(&thread, NULL, threadFunc, (void *) t);  //create a thread
  if(rc) {
    printf("Error; return code from pthread_create() is %d\n", rc);
    exit(-1);
  }
  rc = pthread_join(thread, &reBuf);  //wait for the specified thread to exit
  if(rc) {
    printf("Error; return code from pthread_join() is %d\n", rc);
    exit(-1);
  }
  //print the value returned from thread
  printf("Return value: %s\n", (char *) reBuf);
  return 0;
}
```

# Thread Example (cont.)

```
root@3ca591a8c018:/var/myLibrary/Linux/thread# gcc -w main.c -pthread
root@3ca591a8c018:/var/myLibrary/Linux/thread# ./a.out
--------------------------------
This is Thread Function
Thread ID: 139819801200384
Argument: 100
User input: This is thread example code!
--------------------------------
Return value: This is thread example code!
```

- While making pthread program, ensure to add "-pthread" option to command line. (Link libpthread.a library)

# Other API

- **sleep()**
  - sleep() function is provided by unistd.h library which is short cut of Unix standard library.
  - sleep() function will cause the current executable (a thread or process) to sleep for a period of specified time.

```
#include<unistd.h>
```

```
unsigned int sleep(unsigned int time);
```

```
EX: sleep(10);
```

time**:** how long do you want to sleep the thread. (unit: second)

# Other API

- **rand()**
  - rand() function is provided by stdlib.h library.
  - Returns a pseudo-random number in the range of 0 to RAND_MAX.

```
#include<stdlib.h>
```

```
int rand(void);
```

```
EX: rand();
```

# Other API

- **srand()**
  - srand() function is provided by stdlib.h library.
  - This function seeds the random number generator used by the function rand().

```
#include<stdlib.h>
```

```
void srand(unsigned int seed);
```

```
EX: srand(1000);
```

# Other API

- **time()**
  - time() function is provided by time.h library.
  - Calculates the current calender time and encodes it into time_t format.

```
#include<time.h>
```

```
time_t time(time_t *t);
```

```
EX: time(NULL);
```

# Random number example

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main()
{
 int a;
 srand(time(NULL));
 for(int i=0; i<=5; i++){
            a=(rand()%100)+1;
            printf("The Random Number is %d \n", a);
 }
}
```

```
arthur@arthur-VirtualBox:~/hw2$ ./rand
The Random Number is 14
The Random Number is 87
The Random Number is 7
The Random Number is 27
The Random Number is 98
The Random Number is 37
```

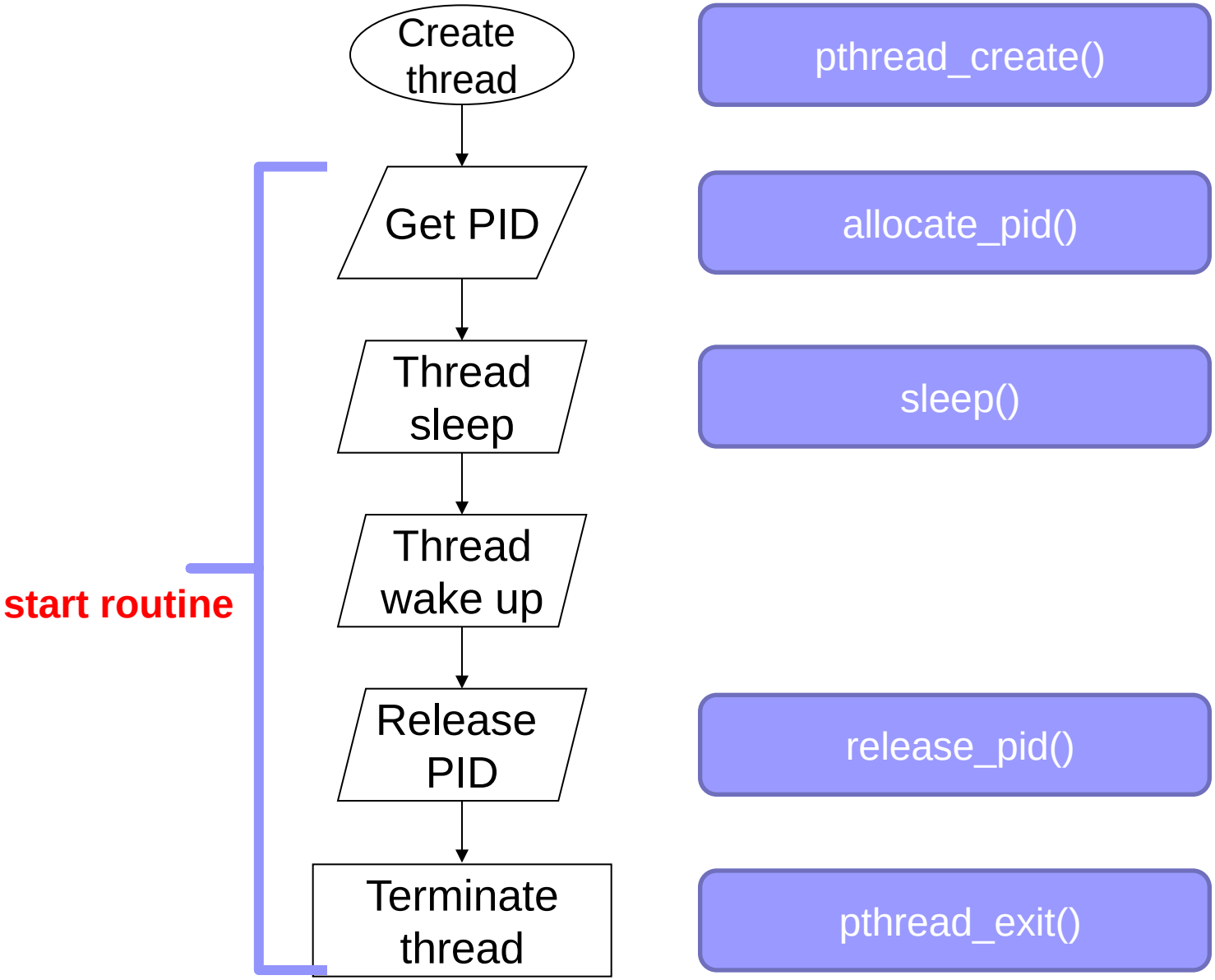
# Outline

- Thread
- Application Programming Interface
  - Exercise 3.20 API
  - Pthreads API
  - Other API
- **Homework Assignment #2**
- Race Condition
- Reference

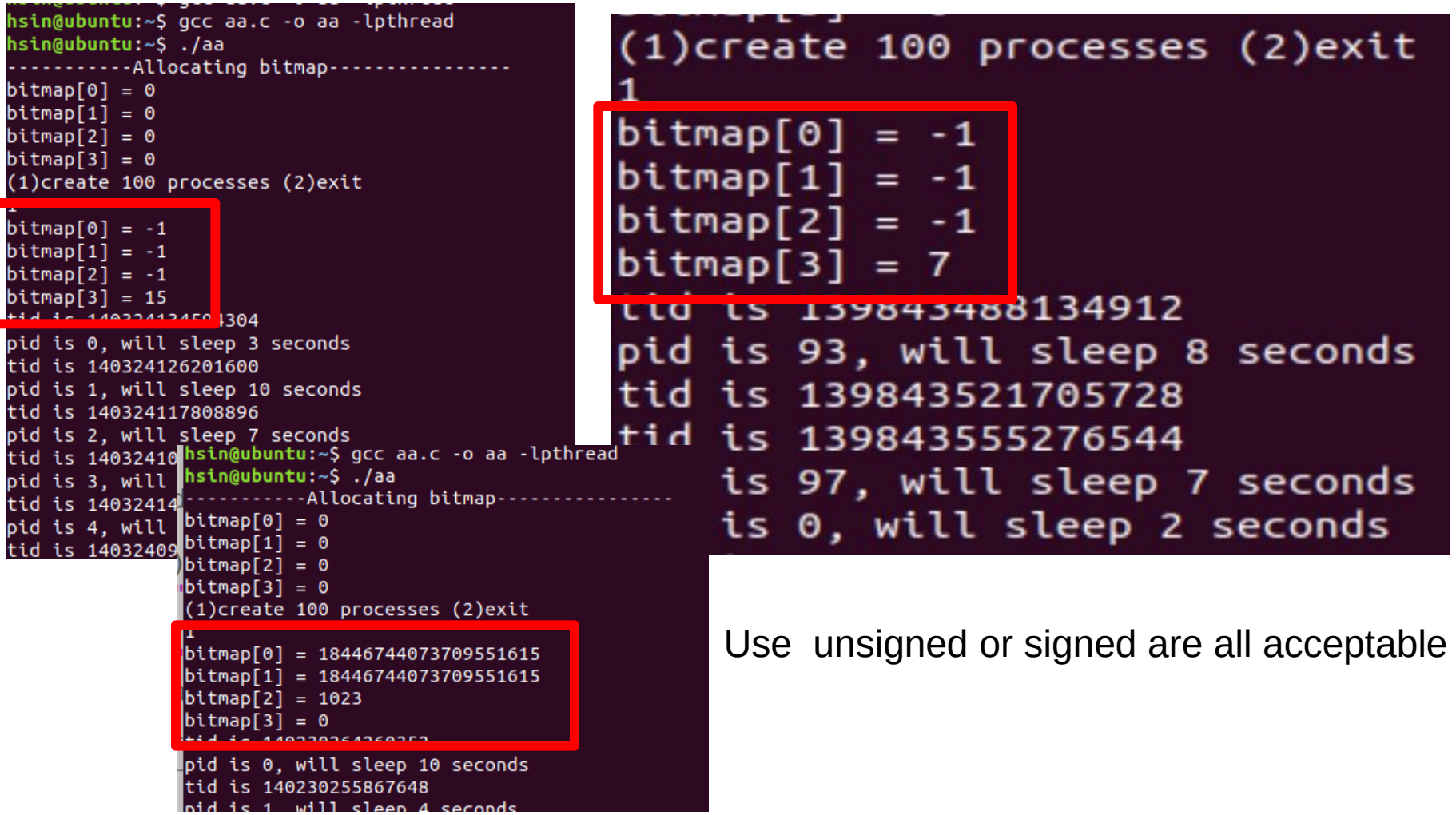
# Homework Assignments #2

- Create 100 threads.
- For each thread, perform the following steps in the start routine
  - First step: Assign PID for the thread.
  - Second step: Print out the PID of the thread.
  - Third step: Let thread sleep for a random period by using sleep() function.
  - Fourth step: Release the PID of the thread, when the thread wake up.
  - Fifth step: Terminate the thread.

# Thread Flowchart



# Result

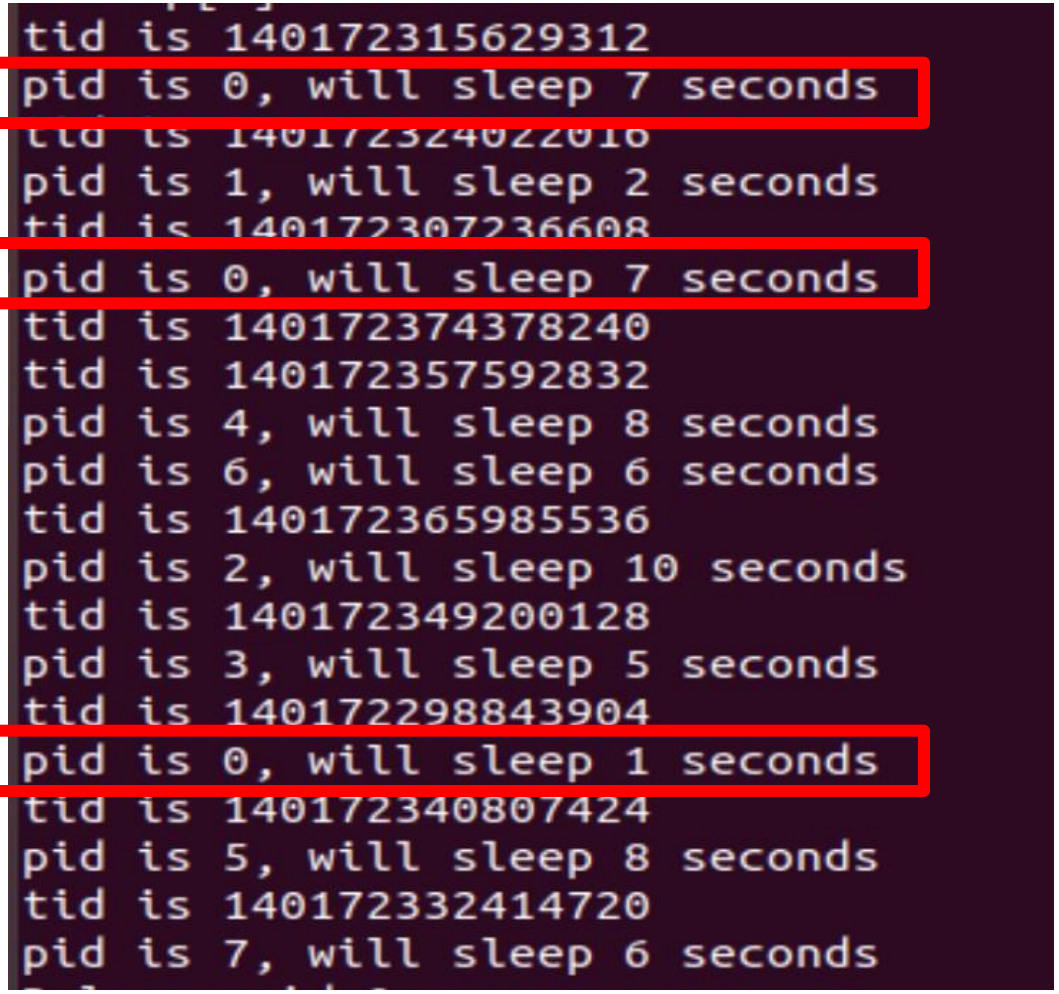



Use unsigned or signed are all acceptable

# Result

```
Release pid 66
Release pid 81
Release pid 73
Release pid 93
Release pid 95
Release pid 1
Release pid 5
Release pid 10
Release pid 17
Release pid 31
Release pid 63
Release pid 94
bitmap[0] = 0
bitmap[1] = 0
bitmap[2] = 0
bitmap[3] = 0
(1)create 100 processes (2)exit
2
Exit this Process !
hsin@ubuntu:~$
```

# Race Condition



# Turn in

- Deadline

  2022/11/17 PM.11:59:59
- Upload to ilearning
- File name
  - HW2_ID.zip (e.g. HW2_7105056035.zip)
    - Source code
      - .c file
    - Word

- If you don’t hand in your homework on time, your score will be deducted 10 points every day.

# Rules

- **No cheat work** is acceptable
  - You get zero if you copy other people's version
- **Only single job** is accepted

# Reference


- https://www.tutorialspoint.com/c_standard_library/index.htm
- http://tw.gitbook.net/c_standard_library/20130920395.html
- https://blog.gtwang.org/programming/pthread-multithreading-programming-in-c-tutorial/
- Operating System Concepts, 10th Edition

# TA

- Name：Hsin-Yu Lin
- Email：sa80419@gmail.com

- Name：Yue-Han Lai
- Email：zcc890408@gmail.com

- Lab: OSNET(1001)